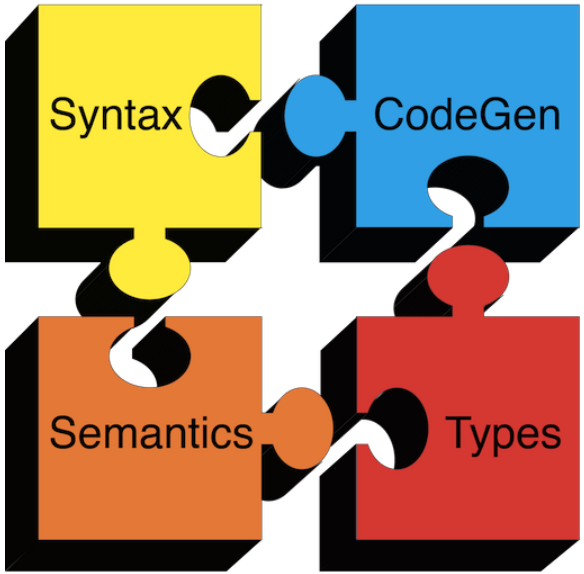

Compilers

Left Recursion

Left Recursion

- Consider a production $S \rightarrow S\ a$

  bool S_1() { return S() && term(a); }

  bool S() { return S_1(); }

- S() goes into an infinite loop

- A left-recursive grammar has a non-terminal S

  $S \rightarrow^+ S\alpha$ for some α

- Recursive descent does not work in such cases

Left Recursion

- Consider the left-recursive grammar

$$S \rightarrow S\ \alpha \mid \beta$$

- S generates all strings starting with a β and followed by any number of α's

- Can rewrite using right-recursion

$$S \rightarrow \beta\ S'$$

$$S' \rightarrow \alpha\ S' \mid \varepsilon$$

Left Recursion

- In general

$$S \rightarrow S\,\alpha_1 \mid \ldots \mid S\,\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$$

- All strings derived from S start with one of $\beta_1,\ldots,\beta_m$ and continue with several instances of $\alpha_1,\ldots,\alpha_n$
- Rewrite as

$$S \rightarrow \beta_1\,S' \mid \ldots \mid \beta_m\,S'$$

$$S' \rightarrow \alpha_1\,S' \mid \ldots \mid \alpha_n\,S' \mid \varepsilon$$

Left Recursion

- The grammar

$$S \rightarrow A\ \alpha \mid \delta$$

$$A \rightarrow S\ \beta$$

  is also left-recursive because

$$S \rightarrow^{+} S\ \beta\ \alpha$$

- This left-recursion can also be eliminated

- See Dragon Book for general algorithm

Left Recursion

Choose the grammar that correctly eliminates left recursion from the given grammar: $E \rightarrow E + T \mid T$

$T \rightarrow id \mid (E)$

- $E \rightarrow E + id \mid E + (E) \mid id \mid (E)$
- $E \rightarrow TE'$
  $E' \rightarrow + TE' \mid \varepsilon$
  $T \rightarrow id \mid (E)$
- $E \rightarrow E' + T \mid T$
  $E' \rightarrow id \mid (E)$
  $T \rightarrow id \mid (E)$
- $E \rightarrow id + E \mid E + T \mid T$
  $T \rightarrow id \mid (E)$

Left Recursion

- Recursive descent
  - Simple and general parsing strategy
  - Left-recursion must be eliminated first
  - … but that can be done automatically

- Used in production compilers
  - E.g., gcc